The Design of a Fault-Tolerant COTS-Based Bus Architecture for Space Applications*

Savio N. Chau Leon Alkalai
Center for Integrated Space Microsystems
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

Ann T. Tai
IA Tech, Inc.
10501 Kinnard Avenue
Los Angeles, CA 90024

Abstract

The high-performance, scalability and miniaturization requirements together with the power, mass and cost constraints mandate the use of commercial-off-the-shelf (COTS) components and standards in the X2000 avionics system architecture for deep-space missions. In this paper, we report our experiences and findings on the design of an IEEE 1394 compliant fault-tolerant COTS-based bus architecture. While the COTS standard IEEE 1394 adequately supports power management, high performance and scalability, its topological criteria impose restrictions on fault tolerance realization. To circumvent the difficulties, we derive a "stack-tree" topology that not only complies with the IEEE 1394 standard but also facilitates fault tolerance realization in a spaceborne system with limited dedicated resource redundancies. Moreover, by exploiting pertinent standard features of the 1394 interface which are not purposely designed for fault tolerance, we devise a comprehensive set of fault detection mechanisms to support the fault-tolerant bus architecture.

Keywords: COTS, IEEE 1394, space applications, bus network reliability, fault-tolerant bus architecture

Submission Category: Regular Paper

Principal Contact: Leon Alkalai, Leon.Alkalaj@jpl.nasa.gov

*The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

1 Introduction

Starting in FY 98, NASA's Office of Space Science has initiated the Advanced Spacecraft Systems Development Program, also known as X2000, to develop advanced spacecraft technologies for future deep-space exploration missions. One of the focus technology development areas is advanced avionics technologies being developed at the newly established Center for Integrated Space Microsystems (CISM), a Center of Excellence at NASA's Jet Propulsion Laboratory [1]. The main focus of CISM is the development of highly integrated, reliable, and highly capable micro-avionics systems for deep-space, long-term survivable, autonomous robotic missions.

The X2000 Program is aimed at delivering a new generation of spacecraft systems every three years, to real flight projects (that is, to the missions) [2]. Currently, there are at least five flight projects that have been identified as direct customers of the X2000 First Delivery technologies in the year 2000, namely,

Europa Orbiter will orbit the moon of Jupiter that has recently been imaged by the Galileo spacecraft. The high radiation environment is a challenge, requiring special electronics design, and extensive radiation shielding. Reduction of mass is critical to the mission success.

Pluto/Kuiper Express will be a mission to image the planet Pluto and go beyond to explore the Kuiper Belt. Long-term survivability, low-power, and autonomous operations are the challenges.

Solar Probe will perform science measurements heading directly to the Sun, within several solar radii. Operating through extreme temperature environments and radiation is the challenge.

Champollion will rendezvous with a comet, land on its nucleus, and sample the comet, performing in-situ measurements. Advanced miniaturization is essential.

Mars Sample Return is a mission engaged by NASA in a coordinated international multiyear robotic exploration of Mars, with the goal of returning samples to Earth. Reduction in mass is essential, as well as on-board autonomous operations.

The earliest launch of the above listed missions is in 2003 (Europa and Kuiper Express). The target missions for the Second Delivery are currently being considered. One should also note that most of the technologies being developed by X2000 are also applicable to Earth orbiting missions. Since the goal of the X2000 Program is to develop multi-mission spacecraft systems technologies for flight projects, the main challenge was to define a scalable, open architecture that can address different requirements (which are often conflicting) such as radiation, temperature, mission complexity, mass, power and volume constraints [3]. Among other things, the most severe constraint is the overall cost of the missions.

With the successful Mars Pathfinder landing on Mars on July 4th 1997, NASA has entered a new era of faster, better, cheaper space exploration (at $150 million, less than some Hollywood movie productions such as Titanic). Under stringent cost constraints, Pathfinder used many commercially available or Commercial Off The Shelf (COTS) technologies. However, while the Mars Pathfinder mission was designed for a 30-day primary mission success (it actually lasted several months), the deep-space missions targeted by the X2000 Program must survive up to 15 years (e.g., Pluto/Kuiper Express).

In this paper, we report in detail our current work at CISM on the design of a distributed, scalable, fault-tolerant multi-mission avionics architecture based on COTS technology (which is referred to as "X2000 architecture" in the remainder of the paper). The architecture is currently the baseline for the Europa Orbiter and Pluto/Kuiper Express projects, both scheduled for launch in the year 2003 [4]. In the X2000 architecture, the multiple computing nodes and devices are symmetric, which means that the roles of computing nodes are interchangeable while devices are treated as intelligent nodes in the network. Moreover, they share a common redundant bus architecture. Most notably, all interfaces used in this distributed architecture are based on COTS. Specifically, the local computer bus is the Peripheral Component Interface (PCI) bus [5]; the "system bus" is the IEEE 1394 high-speed bus [6, 7]; and the engineering bus is the I2C bus [8]. Using strictly COTS Intellectual Property (IP) for all component interfaces is a crucial step toward significant reduction of both system development cost and target cost of the developed system (recurring cost), as COTS interfaces enable other COTS products and IPs to be accommodated by the architecture [2]. The real challenge is to deliver a highly reliable and long-term survivable system based on such an architecture, where the COTS IPs are not developed for mission-critical applications. The spirit of our solution is to maximize the use of standard features of a COTS product in an innovative manner to circumvent its shortcomings, though these standard features may not be originally designed for highly reliable systems.

In the following section, we provide more information about the baseline X2000 architecture, the concept of using COTS in the context of X2000 Program, and the features and disadvantages of IEEE 1394 we exploit and circumvent, respectively, in implementing a fault-tolerant bus architecture. In Section 3, we elaborate the *stack-tree* topology which is IEEE 1394 compliant and exploits IEEE 1394's port-disable feature for bus network reliability. In Section 4, we describe our fault detection mechanisms that support the fault-tolerant bus architecture. Section 5 presents the methods and results of reliability evaluation for the stack-tree topology based bus network. In the concluding remark, we summarize what we have accomplished in this effort and discuss our findings.

2 X2000 Baseline Architecture: A COTS-Based Approach

The proposed baseline X2000 First Delivery avionics architecture is shown in Figure 1, which covers all spacecraft avionics functions including: 1) on-board spacecraft commanding and operations, 2) power management and distribution, 3) science data storage and on-board science processing, 4) telemetry collection, management and downlink, 5) spacecraft navigation and control, 6) autonomous operations for on-board planning, scheduling, autonomous navigation fault-protection, isolation and recovery, etc., and 7) interfacing to numerous device drivers — both "dumb"and "intelligent" device drivers.

The X2000 is a distributed, symmetric architecture with multiple computing nodes and real-time devices connected by a reliable and redundant set of buses. All of the buses that are being used are based on COTS IPs which have been competitively procured. This approach is driven and justified by the requirements of cost reduction for the total system and system development. The COTS buses provide a system level interface to both low-bandwidth (dumb) devices, as well as intelligent devices with embedded micro-controllers.

Further, each computing node consists of: 1) a high-performance processor module (high-performance

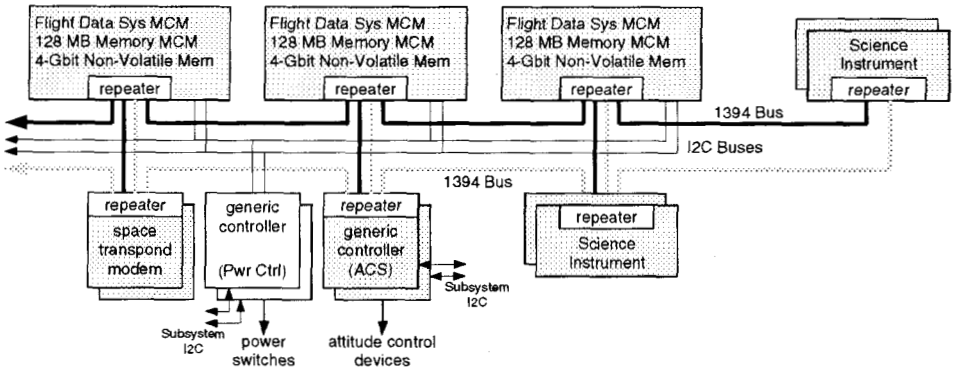

Figure 1: Baseline X2000 First Delivery Avionics Architecture

for space applications implies a speed around 100 MIPS), 2) a 128 Mbytes of local (DRAM) memory, and 3) a 128 Mbytes of on-board non-volatile storage for critical spacecraft information as well as science data. All of these modules communicate via an inter-module 33 MHz PCI bus. The I/O module also provides for the redundant IEEE 1394a interface to other computer nodes and device drivers. The same I/O module also provides the I2C interface which is a low-bandwidth engineering bus.

All the computing nodes over the 1394 bus can be used in a symmetric fashion to control the on-board spacecraft functions. Moreover, the computer redundancy will be exploited for additional on-board capabilities such as fault-tolerant operations, dynamic fault-detection, on-board software verification for software upgrades. Many of the on-board functions in the distributed architecture will be used at the discretion of the target missions based on available power constraints, mission specific requirements, etc.

2.1 Concept of COTS in the Context of X2000 Architecture

As the term "COTS" has a number of different interpretations, it is important to briefly elaborate what we do and what we do not mean by COTS in the context of X2000 architecture. Some interpretations of COTS for space applications imply the direct use of commercial parts, components, or systems. This was certainly the case in Mars Pathfinder where commercial DRAMs were used in the flight computer, and a commercial modem was used as part of the communication system with the Sojourner Rover. In the X2000 architecture, the term COTS has a unique interpretation. In particular, since at least one of the target X2000 customers, namely, Europa, requires the tolerance of high-radiation environments, all the critical electronic components have to be fabricated on specialized semiconductor foundries. Therefore, for the X2000 architecture, we have decided to "procure" COTS IPs for all inter-component interfaces, which in turn, enables other COTS products and COTS IPs to be incorporated into the architecture. While the IPs are COTS products, the actual fabrication of chips and other components are basically carried out by radiation hardened foundries. In that sense, the actual components are COTS IP based and specialized for space use, while the actual interfaces, protocols, etc. are all COTS standard compliant. With this approach, we will reap the benefits of COTS, namely, lower cost of system development, test and integration, as well as lower target recurring cost, while meeting the radiation requirements of our target missions.

2.2 Rationale for Selection of IEEE 1394 Bus Architecture

In the process of selecting the high-speed and low-power buses, many commercial interfaces have been examined. The candidates for the high-speed bus included the IEEE 1394, Fiber Channel, Universal Serial Bus (USB), Fast Ethernet, Serial Fiber Optic Data Bus (SFODB), ATM, Myrinet, FDDI, AS1773, and SPI. Many of these buses (e.g., USB, AS1773, and SPI) fail to meet a projected requirement of 40 Mbps. Others have high power consumption which is unacceptable by deep space applications (e.g., Fiber Channel, SFODB, ATM, and Myrinet). Some of them are not suitable for real-time applications because of the indeterminacy of bus latency. Another important consideration is that the bus should have either radiation-hardened components or an ASIC core design that is portable to a rad-hard foundry. A rigid evaluation based on these factors results in the selection of the IEEE 1394 bus.

Similar criteria were given to the low-power bus selection with special emphasis on low-power consumption and much less consideration for performance. The candidates included the I2C, Controller Area Network (CAN), J1859, Low Power Serial Bus (LPSB, a 1553 Bus modified for low power), MicroLAN, and Access Bus. Our trade study shows that the I2C is the best compromise.

The 1394 and I2C are not the ideal buses from the traditional fault tolerance point of view. Although the 1394 bus has some fault detection features, its fault isolation capability is mediocre and it does not directly provide us with fault recovery mechanisms such as built-in redundancy and cross-strapping. Moreover, IEEE 1394 mandates a tree topology that is in general vulnerable to network partitioning. Nonetheless, our tradeoff study justifies the selection of these two buses because of their low cost and substantial commercial support. The selection of 1394 and I2C enables the X2000 Program to procure COTS ASIC core designs, which can be integrated into a single chip. It is estimated that this approach will reduce the design effort by 30% when compared with the Cassini ASIC design, while the complexity of the ASIC is increased by 400%. Moreover, COTS products required by IEEE 1394 and I2C implementation, such as bus monitors, prototype boards, and device drivers are readily available, which in turn, leads to further big savings.

2.3 IEEE 1394: Pertinent Features and Restrictions for Fault Tolerance

The IEEE 1394 bus is intended to be used for commercial applications such as multimedia and portable phones. The current version of the IEEE 1394 bus can support data rates of 100 Mbps, 200 Mbps, and 400 Mbps for the *cable implementation*, and 50 Mbps and 100 Mbps for the *backplane implementation*. Higher data rates will be attainable in the forthcoming IEEE 1394b. We have selected the cable implementation due to its extensive commercial support. Accordingly, unless it is explicitly stated, all discussions in this paper refer to the cable implementation.

Since the IEEE 1394 bus is designed for real-time multimedia applications, special attention has been paid to guarantee that data can be delivered in time. Hence, the IEEE 1394 bus implements a technique called *isochronous transaction*. All the nodes requiring on-time delivery are called isochronous nodes. Once every 125 μs (an isochronous cycle), each isochronous node has to arbitrate but is guaranteed a time slot (allocated bus bandwidth) to send out its isochronous messages. At the beginning of each isochronous cycle, the root sends out a cycle start message and then the isochronous transaction will follow. Within each

isochronous cycle, 80% of the time is available to the isochronous transactions. The protocol of the IEEE 1394 bus is shown in Figure 2.

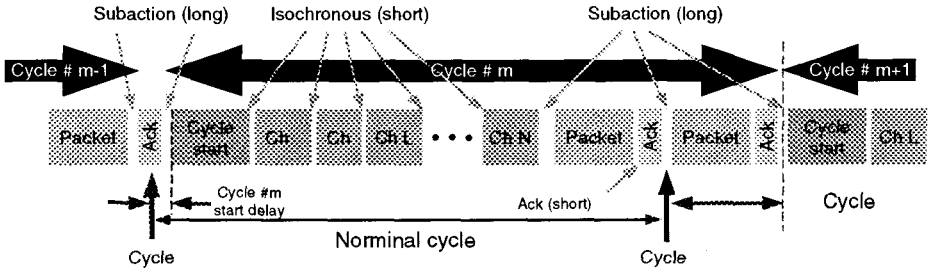

Figure 2: IEEE 1394 Protocol

While the isochronous cycles guarantee bandwidth and *tightly bounded bus latency*, it does not assure reliable delivery since no acknowledgment is required. On the other hand, asynchronous transactions require acknowledgment and therefore can guarantee reliable delivery. However, the bandwidth of asynchronous transaction is not guaranteed because it is allotted only 20% of the isochronous cycle, while many nodes may be arbitrating for that time slot. To avoid starving nodes, the asynchronous transaction employs a fair arbitration scheme, so that every node can send message only once in each fair arbitration cycle. A fair arbitration cycle can span over many isochronous cycles, depending on how much of each cycle is used up by isochronous transactions and how many nodes are arbitrating for asynchronous transactions. The end of a fair arbitration cycle is signified by an arbitration reset gap. As described in Section 4, in implementing a fault-tolerant bus architecture, we exploit the characteristics of the protocol such as gap timing for fault detection and isolation.

As mentioned earlier, the cable implementation of IEEE 1394 mandates a tree topology. Although there are various types of tree structure, for space applications, it is preferred to have a "regular" topology (in the sense that the topological structure can be easily maintained as nodes are added or deleted from the system) because it can simplify the test and integration processes for substantial cost saving. Therefore, the stack-tree topology depicted in Figure 3 is proposed, where a node is either a flight computer or a device. There are three physical layer ports in each node. For each stem node, two or more of these ports are connected to the other nodes, while a leaf node has only one port connected. Furthermore, each connection in Figure 3 is actually two twisted wire pairs, referred to as TPA/TPA* and TPB/TPB* ("*" symbolizes the complement signal). The TPA and TPB signals are designed for arbitration, data transmission, node insertion/removal detection, and indication of node data rate. We take advantage of this standard feature in designing the detection mechanisms for certain bus failure modes, such as babbling nodes (described in Section 4).

During bus startup or reset, the bus will go through an initialization process through which each node will get a physical node ID. In addition, the root (cycle master), bus manager, and isochronous resource manager will be elected. The root mainly is responsible for sending the cycle start message and acts as the central arbitrator for bus requests. The bus manager is responsible to acquire and maintain the bus topology. The isochronous resource manager is responsible for allocating bus bandwidth to isochronous nodes. The root, bus manager, and isochronous resource manger are not fixed, so that any qualified nodes can be elected

to take these roles when needed. Clearly, this dynamic initialization feature can be utilized to support bus network reconfiguration.

Once the bus initialization is complete, the bus will enter the normal operation. In either the isochronous or asynchronous mode, any node wishes to send data must arbitrate for the bus. The arbitration is carried out by the two twisted wire pairs TPA and TPB. A useful feature worth to mention is that the signaling state (TPA, TPB) used in bus arbitration contains comprehensive information about the status of the nodes and the bus network, which can be used for fault monitoring.

Note that the stack-tree topology shown in Figure 3 has a potentially serious drawback. That is, a tree topology by itself is not fault tolerant as any single link failure will partition the tree into two segments and any single node failure can break the tree into three parts. What makes the design more difficult is that *spare nodes dedicated for fault tolerance are not permitted in the X2000 architecture due to power constraint.* Although various schemes of fault-tolerant bus network have been proposed in research literatures (see [9, 10], for example), the restrictions from 1394 and from our application prevent us from utilizing those schemes since the majority of them involve either loops or spare nodes.

There are some fault detection provisions such as CRC in the 1394 standard, but they are inadequate to ensure the reliability required for long-life missions such as Pluto/Kuiper Express (a 12 to 15 year mission). On the other hand, IEEE 1394a [11] provides an employable feature called "port-disable," which allows us to implement a 1394 compliant reconfigurable bus architecture, though this feature is not purposely designed for fault tolerance. The spirit of our solution is to maximize the use of pertinent standard features of the COTS product in question to circumvent its shortcomings, though most of these standard features are not designed for reliability purpose. In the following sections, we describe the design of a COTS-based fault-tolerant bus architecture in detail with respect to bus network topology and fault detection methodologies.

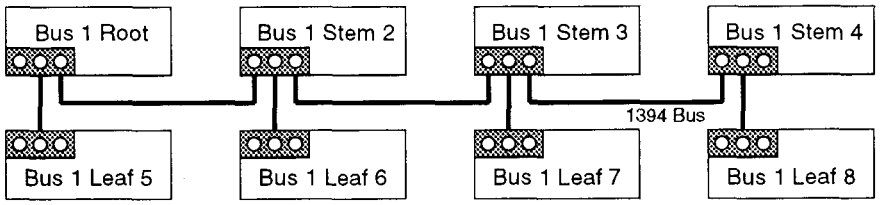

Figure 3: Bus Network based on Stack-Tree Topology

3 Stack-Tree Topology based Bus Architecture

3.1 Concepts

In the interest of bridging the terminology between network topology and the X2000 MCM-stack packaging technology [12], we call the proposed topology "stack-tree topology."

Definition 1 *A stack tree is a tree where each stem node is connected to at most three other nodes among which at most two are stem nodes.*

For example, the trees in Figures 4(a), (c) and (d) are stack trees while that in Figure 4(b) is not.

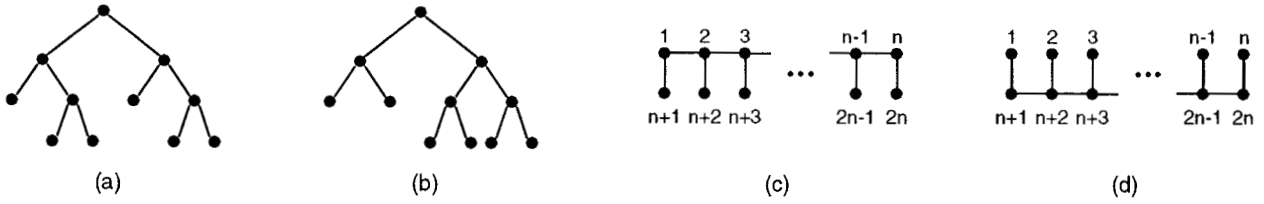

Figure 4: Trees

Definition 2 *A complete stack tree is a stack tree where each stem node is connected to at least one leaf node.*

Figure 4(c) depicts a complete stack tree (CST) with n stem nodes. We call this topology *simplex complete stack tree* which is denoted as CST_S. Note that the nodes are labeled such that the stem nodes have ID numbers from 1 to n, while the leaf nodes have ID numbers from $n+1$ to $2n$. This labeling scheme will be used in the remainder of the paper. Further, we use n, the number of stem nodes in a CST, to denote the *size* of the tree. Note also that the trees in Figures 4(c) and (d) are both CSTs. Based on the CST in Figure 4(c), we can define *CST mirror-image* as follows.

Definition 3 *The mirror-image of a complete stack tree is a tree obtained by (1) removing the edges connecting the stem nodes with ID numbers i and j which satisfy the relation $|i-j|=1$; (2) adding edges to connect the leaf nodes with ID numbers k and l which satisfy the relation $|k-l|=1$.*

Clearly, the CST shown in Figure 4(d) is a mirror image of that in Figure 4(c). It is worth to note that a CST and its mirror image do not have any stem nodes in common. Moreover, based on the above definitions, it can be shown that the mirror-image of a CST is also a CST.

3.2 Applications

The performance of the X2000 spaceborne systems is scalable and gracefully degradable. Accordingly, our objective is to develop a fault-tolerant bus network architecture that will allow all the surviving nodes in the bus network to remain connected in the presence of node failures, without requiring spare nodes. The fact that a CST and its mirror image do not have stem nodes in common implies that losing a stem node in one tree will not partition its mirror image. Accordingly, a dual bus scheme comprising a CST and its mirror image, referred to as *CST dual scheme* (denoted as CST_D), as shown in Figure 5(a), will be effective in tolerating single or multiple node failures given that 1) the failed nodes are of the same type (all stem or all leaf) with respect to one of the CSTs (see Figure 5(b)), or 2) the failed nodes involve both stem and leaf nodes but they form a cluster at either end (or both) of a CST, which will not affect the connectivity of the remainder of the tree (see Figure 5(c)). We use *terminal clustered stem-leaf failures* to refer to the second failure pattern. Thus, for the cases which involve only the above failure patterns, all the surviving nodes will remain connected (no network partitioning). On the other hand, if a stem node and a leaf node in a CST_D based network fail in a form other than terminal clustered stem-leaf failure (see Figure 5(d)), both the primary and mirror image will be partitioned.

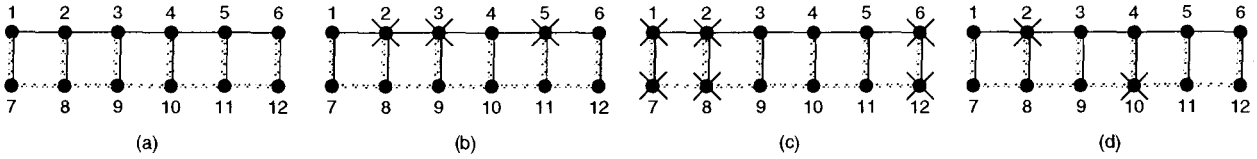

Figure 5: CST-Based Dual Bus Network

With the motivation of building a robust bus network architecture capable of tolerating more node failures (in terms of number and type), we exploit a unique feature of IEEE 1394, namely, the *port-disable* capability [11]. This feature enables the physical connections between the physical layer of a node and the serial bus cable to become "invisible." The implication is the following:

1) By using disabled ports, backup connections between nodes can be added without forming loops (recall that loops are prohibited by IEEE 1394). By "backup connection," we mean a serial bus cable that connects (via disabled ports) two nodes which are not expected to have a direct connection in the original network configuration (differing from duplicated connection); and

2) Upon fault detection, by disabling physical ports, a failed node will be allowed to be isolated from the rest of the bus network, and necessary backup link(s) can be activated (by enabling the corresponding ports) to repair the partitioned network such that messages can be routed in a reconfigured network, bypassing the failed node.

Consider a bus network based on the CST_S topology with n stem nodes, each of which has one leaf node, as shown in Figure 6(a). If we add a backup link between any two leaf nodes labeled i and j which satisfy the relation $|(i \bmod n) - (j \bmod n)| = 1$, and also add a backup link to connect stem nodes 1 and n, then we get a topology as shown in Figure 6(b) (an instantiation of the topology in which $n = 6$). Because the added connections (dashed edges) are of inactive nature, the bus network remains free of loop and thus complies with the 1394 tree topology criterion. Figure 6(c) illustrates the bus network from a 3-dimensional perspective, where the network topology resembles a ring. Accordingly, we denote this bus network configuration as CST_R. To aid the description of failure mechanisms of a CST_R based bus network, we introduce the following terminology:

Definition 4 *A failed stem node i and a failed leaf node j in a CST_R based network of size n will form a cut-type failure if $|(j \bmod n) - (i \bmod n)| \leq 1$.*

Figure 7 illustrates the concepts of cut-type and non cut-type failures. Specifically, the failures comprised by nodes 2 and 9 in Figure 7(a), nodes 5 and 11 in Figure 7(b), and nodes 1 and 12 in Figure 7(c) are cut-type failures. On the other hand, the node failures shown in Figures 7 (d) and (e) are not cut-type failures.

Further, we use the term *clustered failure* to refer to the failure of a group of nodes which are adjacent to each other such that there exists a shortest path between any two failed nodes in the group which does not go across a surviving node. Figures 8(a) and (b) illustrate the scenarios of clustered and non-clustered multiple cut-type failures, respectively. Clearly, while the latter failure pattern leads to a bus network partition,

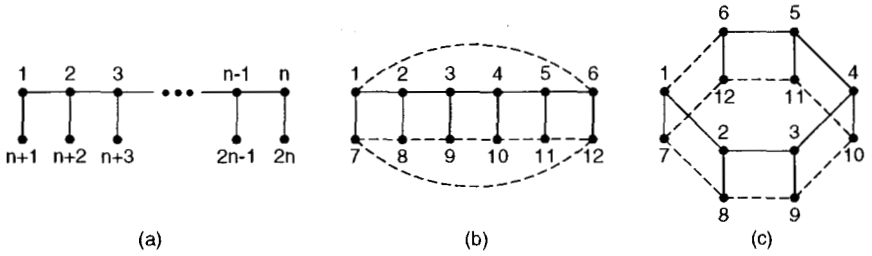

Figure 6: CST-Based Bus Network and Disabled Backup Links

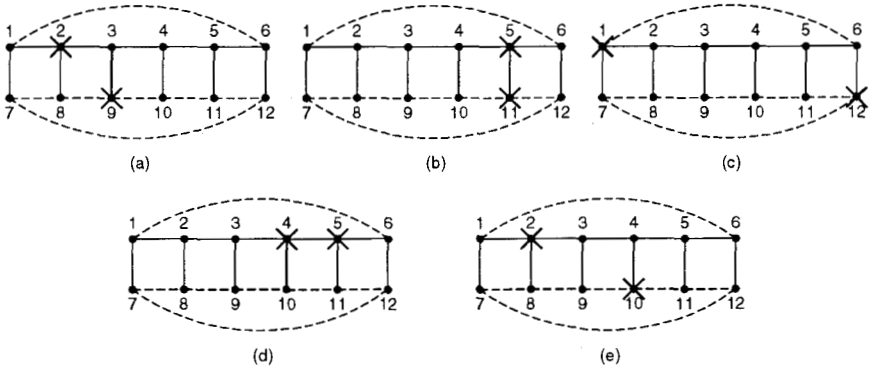

Figure 7: Cut-Type and Non Cut-Type Failure

the former does not even if node 6 also fails, although both scenarios involve multiple cut-type failures. Therefore, *a bus network based on the CST_R topology will be partitioned if and only if there exist multiple cut-type failures which do not constitute a single cluster.* Intuitively speaking, as illustrated by Figures 9(a) and (b), the first cut-type failure (single or clustered) will break the ring structure so that the remainder of the network becomes a CST_S based structure (with backup links); whereas the second cut-type failure (single or clustered) will break the CST_S based structure, resulting in network partitioning (see Figure 9(c)).

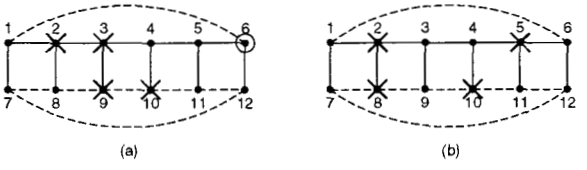

Figure 8: Clustered and Non-Clustered Multiple Cut-Type Failures

Figure 10 depicts the simplified X2000 architecture in which the CST_R based bus network described above is implemented. In the figure, the solid and dashed thick lines marked "1394 Bus" represent the active and backup links, respectively. During normal operation, the active connections are driven by enabled ports while the ports of backup connections are disabled to avoid loops. The thin lines marked "I2C Buses" correspond to the interface for fault detection, isolation and reconfiguration. The I2C bus is a very simple low-speed multi-drop bus and used only for protecting the 1394 bus. Hence this engineering bus has very low utilization and power consumption. For additional protection, a redundant bus (consisting of the 1394 and I2C buses) which is a mirror image of the configuration shown in Figure 10 is proposed by our design.

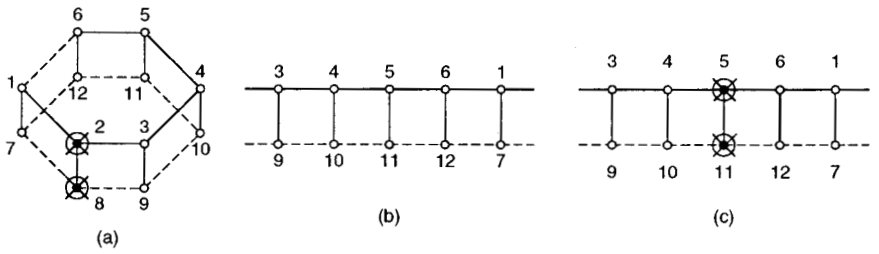

Figure 9: CST_R Based Bus Network Partitioning

For clarity of illustration, the connections of the redundant bus are not shown in the figure.

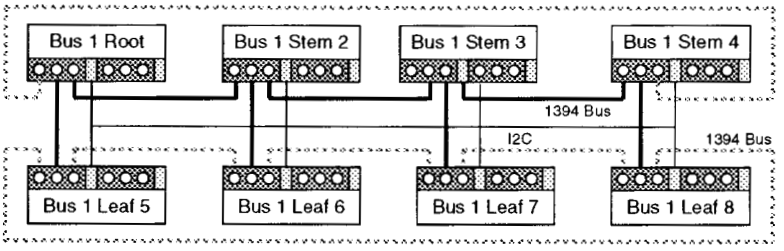

Figure 10: CST_R-Based Fault-Tolerant Bus Network Architecture

4 Devising Fault Detection Mechanisms based on 1394 Standard Features

4.1 Design Principles

Our fault detection, isolation and recovery algorithms for supporting the fault-tolerant bus architecture described in Section 3.2 are driven by the following failure modes which are crucial for space applications: 1) *invalid messages* that contains invalid data detectable by error codes, 2) *no response failure mode* in which an expected response to a message does not return in time, 3) *babbling failure mode* in which uncontrolled data flooding causes normal communication in the bus network to be blocked or interrupted, and 4) *aliasing failure mode* in which multiple nodes wrongly have the same ID.

Based on the coordination between the 1394 and I2C buses (recall that the later is a low-cost engineering bus aimed at assisting fault detection, isolation and recovery in the X2000 architecture), a comprehensive set of fault tolerance strategy can be devised [13]. Our design principle is to first fully utilize the standard error detection primitives available in IEEE 1394, and then exploit its other pertinent features to devise additional fault tolerance mechanisms to compensate inadequacies. Among other fault tolerance methodologies developed for the bus architecture, we focus on fault detection mechanisms in this section. We first present the basic means for fault detection and then describe more specific detection strategies from failure mode perspective. Although the I2C bus plays an important role in most of the algorithms, we omit the detailed description as it is beyond the scope of this paper.

4.2 Basic Means for Fault Detection

4.2.1 Fault Detection Provisions in IEEE 1394

IEEE 1394 provides a number of standard fault detection primitives, including 1) data CRC and packet header CRC for both isochronous and asynchronous transactions, 2) error codes in acknowledgment packets to indicate whether a message is successfully delivered, 3) parity bit to protect the acknowledgment packet itself, 4) error codes in response packets to indicate if the requested action has been completed successfully, and 5) built-in timeout conditions such as response timeout, arbitration timeout, acknowledgment timeout.

When errors are detected by the target node during the asynchronous transaction, the target node returns the acknowledgment packet and response packet with error codes to the requesting node. Based on the error code, the requesting node will carry out user defined fault recovery actions. The error codes specified for the acknowledgment packet correspond to 1) resource contention causing bus request denial, and 2) data error and packet-type error detected by the data CRC and header CRC, respectively. The error codes specified for the response packet correspond to 1) resource contention, 2) data unavailability, 3) unsupported packet type, and 4) unaccessible address location. More detailed description of these fault detection provisions can be found in [6, 7, 11].

4.2.2 Methods of Heartbeat and Polling

We use heartbeat and polling mechanisms to augment the fault detection capability of IEEE 1394. In particular, an isochronous cycle start message (see Section 2.3) sent by the cycle master (root) of the 1394 bus every 125 μs can be utilized as the heartbeat. By letting all the nodes participate monitoring the interval between cycle start messages, heartbeat will be effective for detecting various failures. For example, when the root node fails, other nodes will detect a missing cycle. Another example is the babbling node failure mode. When a node babbles, the cycle start message will be either corrupted or lost, which causes missing heartbeat. Whereas if a fault causing multiple nodes to have the root ID, the heartbeat interval will become shorter and irregular because each "faked root" will send its cycle start message.

The root node also sends polling messages periodically to individual nodes by using asynchronous transaction. Since an asynchronous transaction requires acknowledgment from the target node, a node failure can be detected by acknowledgment timeout. To reduce the polling latency, a unique feature specified by 1394a [11] for performance improvement of asynchronous transactions, called "fly-by concatenation," can be exploited. With fly-by concatenation, when an acknowledgment packet is en route from a node toward the root, any intermediate nodes can attach its own packet to the original packet. Hence, when the root polls a leaf node, all the upstream stem nodes can attach their health status (HSPs) to the acknowledgment packet from the polled leaf node. Figure 11 illustrates this polling mechanism, where the node IDs are based on the architecture shown in Figures 3 and 10. As a result, the root needs only to poll the leaf nodes, which reduces the polling cycle by half. A side benefit of this mechanism is that, the stem nodes, whose error conditions in general have greater impacts to the integrity of the bus, will be exercised more often and thus have shorter fault detection latencies.

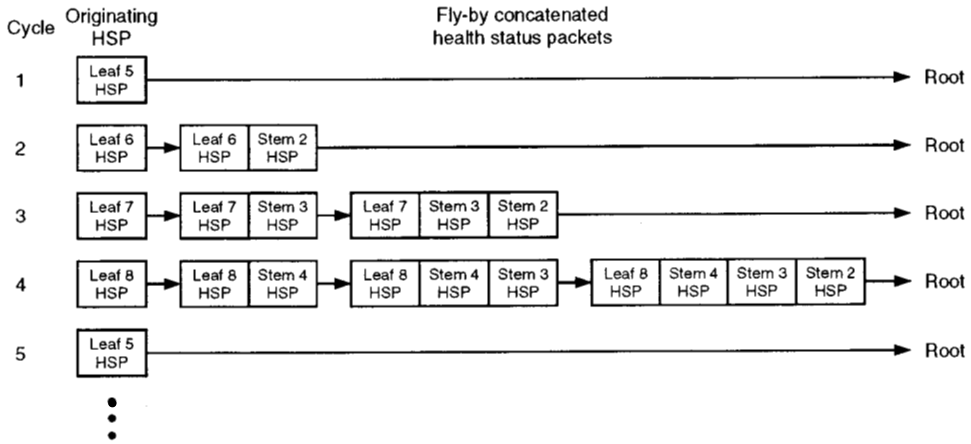

Figure 11: Fault Detection based on Fly-By Concatenation

4.2.3 Watchdog Timers

The detection mechanisms described above are supported by a set of watchdog timers, among which the key timers are:

CPU Watchdog Timer : A hardware timer to monitor the health of the local CPU. This timer is reset by the CPU periodically. If it times out, a local reset will be triggered.

Heartbeat Interval Monitor (HIM) : A hardware timer to monitor the heartbeat intervals. It is reset upon the arrival of a cycle start message.

Heartbeat Lost Timer (HLT) : A hardware timer triggered by the lost of heartbeat. It postpones the recovery action by the local node in order to avoid the scenario in which multiple nodes simultaneously carry out recovery actions.

Poll Response Timer (in Root Node) : A software timer to monitor the response time of polling messages on the 1394 bus.

4.3 Detection Mechanisms: From Failure Mode Perspective

4.3.1 Invalid Messages Failure Mode

When a packet is transmitted across the serial bus, it may get corrupted, which results in invalid data. As data corruption can be detected by CRC or parity bit, the message will be retransmitted. If retry is successful, the normal bus transactions will resume. Otherwise the original requesting node can employ the I2C bus to isolate the fault and notify the root to initiate the reconfiguration process.

4.3.2 No-Response Failure Mode

A number of no-response failures can be detected if they cause violation of the bus gap timings specified in the 1394 standard. For example, if a node fails to return or transmit an acknowledgment packet in an

asynchronous transaction, it can be detected by the acknowledgment gap timeout. On the other hand, for the isochronous transactions which do not require acknowledgment packets, no-response failure will not be detected by gap timing violation. Therefore, if a no-response failure occurs in an isochronous node or its upstream nodes, the failure may go undetected. In that case, the heartbeat and polling mechanisms described in Section 4.2.2 will effectively detect the failure. It is worth to note that since a no-response failure can partition a tree topology based network by blocking the communication between the upstream and downstream nodes (relative to the no-response node), the I2C bus will be deployed to bypass the failed node to carry out reconfiguration process.

4.3.3 Babbling Failure Mode

Babbling failure mode refers the scenario in which a node keep sending data uncontrollably. A babbling node can block all communications in the network and thus results in a serious bus failure. The babbling failure mode can be detected by the sequence of states on the twisted wire pairs (TPA, TPB) (Section 2.3). When a babbling node is present, the normal sequence of arbitration, data prefix, data transfer, and data end will be corrupted. Another detectable form of babbling is a node holding the (TPA, TPB) at the state (1, 1), which causes continuous bus resets. And as mentioned in Section 4.2.2, if the babbling node is the root node, it can be detected through its corrupted or lost cycle start message (the later corresponds to missing heartbeat).

4.3.4 Aliasing Failure Mode

As described in Section 2.3, the physical ID of each node is assigned dynamically during the bus initialization process. When a node ID is corrupted due to a permanent fault or a single event upset such that it coincides with the ID of another node in the network, an aliasing failure occurs.

If the root node has the aliasing problem, it will be detected by the non-root nodes when they attempt to communicate with the root (e.g., for bus arbitration). In particular, upon the detection of the event in which a node sends its message to multiple roots, a bus reset will be triggered by the 1394 protocol. On the other hand, if the aliasing failure occurs in a node other than the root, it can be detected by the polling mechanism described earlier. That is, the root will receive response packets (HSPs) from the two nodes which have the same ID, in responding to the same polling message. Upon the detection, the root can continue its polling process and then identify the faulty node by checking the node IDs marked in the topology map (which is generated during bus initialization).

5 Bus Network Reliability Evaluation

In accordance with the objective of the fault-tolerant bus architecture described in Section 3.2, we define bus network reliability is the probability that, through a mission duration t, the network remains in a state such that all the surviving nodes are connected (no network partitioning). Indeed the causes of a node failure encompass physical layer failure, link layer failure and CPU failure. Moreover, while redundant links (serial

bus cables) are permitted in the X2000 architecture, redundant nodes are not allowed due to the power and weight constraints. As a result, the likelihood of node failure is significantly greater than that of link failure. Therefore, in the reliability assessment that follows, we concern only node failure. Further, we assume perfect coverage for fault detection and reconfiguration mechanisms in this evaluation. Before proceeding to derive solutions of reliability measures, we define the following notation:

$R_{\mathrm{S}}^{\mathrm{CST}}$	The reliability of a $\mathrm{CST_S}$ based bus network.
$R_{\mathrm{D}}^{\mathrm{CST}}$	The reliability of a $\mathrm{CST_D}$ based bus network.
$R_{\mathrm{R}}^{\mathrm{CST}}$	The reliability of a $\mathrm{CST_R}$ based bus network.
λ	Poisson failure rate of a node.
t	Mission duration.
q	Probability that a node fails during a mission.

We begin with analyzing the $\mathrm{CST_S}$ and $\mathrm{CST_D}$ based bus network schemes. As explained in Section 3.2, terminal clustered stem-leaf failures in a CST will not affect the connectivity of the remainder of the tree. Thus we can retrieve a "remainder" from the original CST by eliminating the portion(s) comprised by terminal clustered stem-leaf failures. This observation leads us to solve $R_{\mathrm{S}}^{\mathrm{CST}}$ and $R_{\mathrm{D}}^{\mathrm{CST}}$ as follows. We first condition the reliability of a "remainder" by its size k (the number of its stem nodes) which determines the number of possible positions of the remainder in the original CST, and then we uncondition it. More precisely, there are $(n-k+1)$ possible positions for a remainder of size k in a CST of size n.

Since the reliability of a remainder of size k for a $\mathrm{CST_S}$ based network is the probability of all the stem nodes being failure-free, that is, $(1-q)^k$, we have

$$R_{\mathrm{S}}^{\mathrm{CST}} = \sum_{k=1}^{n} (n-k+1)(1-q)^k q^{2(n-k)} \tag{1}$$

Since the reliability of a size-k remainder in a $\mathrm{CST_D}$ based network is the complement of the probability that at least one stem node and at least one leaf node fail, namely, $\left(1-(1-(1-q)^k)^2\right)$, the measure $R_{\mathrm{D}}^{\mathrm{CST}}$ can be expressed as

$$R_{\mathrm{D}}^{\mathrm{CST}} = \sum_{k=1}^{n} (n-k+1)\left(1-(1-(1-q)^k)^2\right) q^{2(n-k)} \tag{2}$$

In both Equations (1) and (2), $q = 1 - e^{-\lambda t}$ is the probability that a node fails during mission time t with failure rate λ.

The solution for $R_{\mathrm{R}}^{\mathrm{CST}}$ is more difficult because 1) the backup links enable the bus network to tolerate more node failure patterns, which makes the analysis of the conditions that cause network partitioning more complicated, and 2) the ring-like structure makes it difficult to ensure that the failure patterns considered in the model are exhaustive and mutually exclusive.

As explained in Section 3.2, a bus network architecture with a $\mathrm{CST_R}$ topology will be partitioned if and only if there exist multiple cut-type failures which do not constitute a single cluster. In other words, the

surviving nodes in a CST_R based bus network will remain connected if there exists at most one cut-type failure cluster. In the model construction method described below, we view a single cut-type failure as a special case of cut-type failure cluster (where the size of the cluster is one) and treat a network that is free of cut-type failure as a special case of remainder (where the sizes of the cluster and remainder equal to 0 and n, respectively). Specifically, we first condition network reliability by the size of the first cut-type failure cluster (the number of stem nodes involved in the cluster), then we evaluate the probability that the remainder, which is the portion of the CST_R based network structure excluding the first cluster, is free of cut-type failure. This probability is evaluated based on a set of recursive functions that enumerate the combinations and permutations of failed and surviving nodes in the remainder where cut-type failure is absent. By successively expanding and reducing the sizes of the failure cluster and the remainder, respectively, we exhaustively enumerate the probabilities that a remainder is cut-type failure free. Accordingly, the measure we seek to evaluate can be expressed as

$$R_\text{R}^\text{CST} = \sum_{m=0}^{n-1} \binom{n}{m} q^{2m} F(n-m) + 2(n-1)G(n) \tag{3}$$

where the index m represents the size of a cluster, $\binom{n}{m}$ is the number of possible positions of the cluster in the CST_R based network, q^{2m} is the probability of such a cluster, and $F(n-m)$ evaluates the probability that the remainder is cut-type failure free given that the size of the failure cluster is m. This probability is solved by a set of recursive functions which "walk through" the remainder backward, ensuring that 1) the distinct scenarios characterized by the number and positions of failed nodes are exhaustively enumerated, and 2) the remainder is cut-type failure free. More succinctly,

$$F(k) = F_0(k) + F_1(k) + F_2(k) \tag{4}$$

where $F_0(k)$, $F_1(k)$ and $F_2(k)$ are the recursive functions that evaluate the probability of a cut-type failure free remainder of size k, in which 1) both k and $k+n$ are surviving nodes, 2) k is a failed node and $k+n$ is a surviving node, and 3) k is a surviving node and $k+n$ is a failed node, respectively. The corresponding expressions are as follows:

$$\begin{aligned} F_0(i) &= [F_0(i-1) + F_1(i-1) + F_2(i-1)]\,(1-q)^2 \\ F_0(1) &= (1-q)^2 \end{aligned} \tag{5}$$

$$\begin{aligned} F_1(i) &= [F_0(i-1) + F_1(i-1)]\,q(1-q) \\ F_1(1) &= q(1-q) \end{aligned} \tag{6}$$

$$\begin{aligned} F_2(i) &= [F_0(i-1) + F_2(i-1)]\,(1-q)q \\ F_2(1) &= (1-q)q \end{aligned} \tag{7}$$

To aid further explanation of the model, we introduce the term *cut-type failure node pair*, abbreviated as *CFP*, which refers to the failed node pair that forms a cut-type failure. Per Definition 4 in Section 3.2, a stem node k potentially could be involved in three differing CFPs, as shown in Figure 12. Further, we call the CFPs $\{k, k+n-1\}$, $\{k, k+n\}$ and $\{k, k+n+1\}$ backward CFP, vertical CFP and forward CFP, respectively.

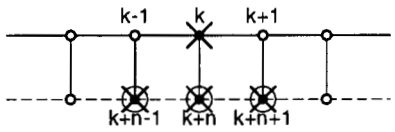

Figure 12: Cut-Type Failure Node Pair

It is worth to note that, due to different patterns of the junctions between the cut-type failure cluster and the remainder, the first term of Equation (3) takes into account not only for the cut-type failure clusters that are constituted by vertical CFPs but also the cut-type failure clusters that can be viewed as the clusters formed by forward and backward CFPs, as illustrated in Figures 13(a) and (b).

For the special case where $m = 0$, the cut-type failure cluster becomes degenerate while the remainder spans the entire CST_R based bus network. Although the combinatoric coefficient $\binom{n}{m}$ equals to unity when $m = 0$, different starting positions of the "remainder" in the CST_R based bus network are still taken into account by the set of recursive functions. That is, Equations (5), (6) and (7) together enumerate the probabilities of all possible sequences of the status of node pairs $\{k, k+n\}$ such that there is no cut-type failure formed within the remainder. As a result, different positions of a "remainder" in a CST_R structure (where $m = 0$) are "inherently" considered by $F(n)$. For example, the "remainder" in Figure 14(b) can be viewed as a result of shifting the starting position of that in Figure 14(a) toward right by one node position — both cases (where $m = 0$) are enumerated by $F(n)$.

The only exception is the scenario in which $m = 0$ and a single cut-type failure (a forward or backward CFP) is formed at the "junction" where the two end of the remainder "merge," as illustrated in Figure 13(c) (recall that a single CFP will not partition the network). Since the recursive functions are formulated in a way such that the cases where the remainder has an internal CFP are excluded, what missed in the first term of Equation (3) but compensated by its second term are the $(n-1)$ different positions of this particular CFP. The coefficient 2 for the second term is necessary because the CFP could be of either a forward or backward type and the two types of CFP are "symmetric" with respect to structure and probability formulation. $G(n)$ evaluates the probability that there exists a single (forward or backward) CFP in the CST_R based network. The derivation of $G(n)$ is based on another set of recursive functions, which are formulated in a manner such that only the scenarios where the "merge" of the ends of the remainder results in a forward or backward CFP are considered:

$$G(n) = [G_0(n-l) + G_1(n-l)]\, q(1-q) \tag{8}$$

$$G_0(i) = [G_0(i-1) + G_1(i-1) + G_2(i-1)]\, (1-q)^2 \tag{9}$$

$$G_0(1) = 0$$

$$G_1(i) = [G_0(i-1) + G_1(i-1)]\, q(1-q) \tag{10}$$
$$G_1(1) = 0$$

$$G_2(i) = [G_0(i-1) + G_2(i-1)]\, (1-q)q \tag{11}$$
$$G_2(1) = (1-q)q$$

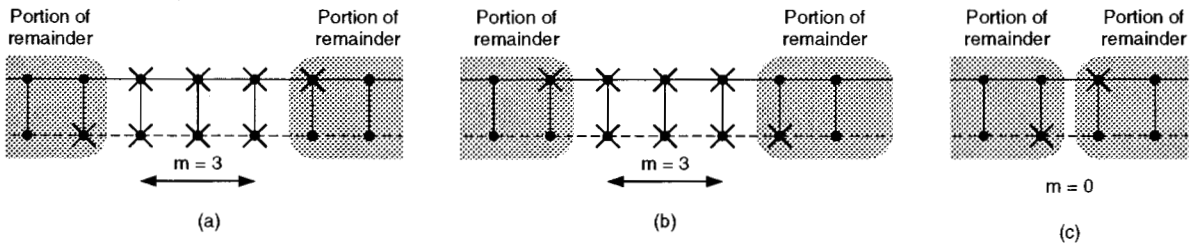

Figure 13: Failure Cluster and Remainder

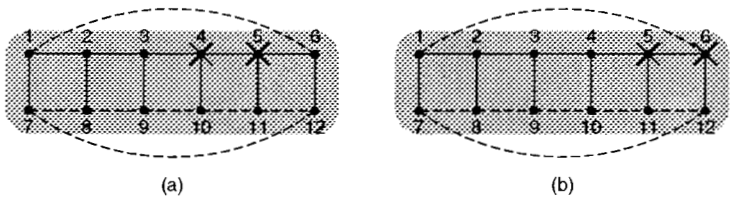

Figure 14: "Positions" of a Remainder of Size n

Applying the models described above, reliability measures for the bus networks based on topologies CST_S CST_D and CST_R are evaluated with respect to the node failure rate λ, size of a bus network n and mission duration t (in hours). Figure 15 depicts R_S^CST, R_D^CST and R_R^CST as functions of component node failure rate λ. In this evaluation, the size of the CST-based bus networks n is set to 16 (a 32-node network) and mission duration t is set to 90,000 hours which implies an over 10-year long-life mission. It can be observed from the figure that, while CST_D results in an appreciable amount of improvement from CST_S, CST_R leads to significantly more reliability gain. The quantitative results show that R_R^CST can reach 0.9999999762 if $\lambda = 10^{-8}$ and nearly perfect reliability can be achieved if λ is 10^{-9} or lower.

Figure 16 shows the results of the evaluation in which λ is set to 10^{-7} and t is set to 90,000 hours, while n is kept as a variable parameter. It is interesting to note that R_D^CST equals to R_R^CST when $n = 2$. This is a reasonable result because for a 4-node network, the node failure patterns that will partition a CST_D based network coincide with the failure patterns that will partition a CST_R based network. It can also be observed that the reliability improvement by R_R^CST from R_D^CST becomes more significant as the size of the network increases. This is because more routing alternatives (comprised by active and backup links) are available in a larger CST_R based network.

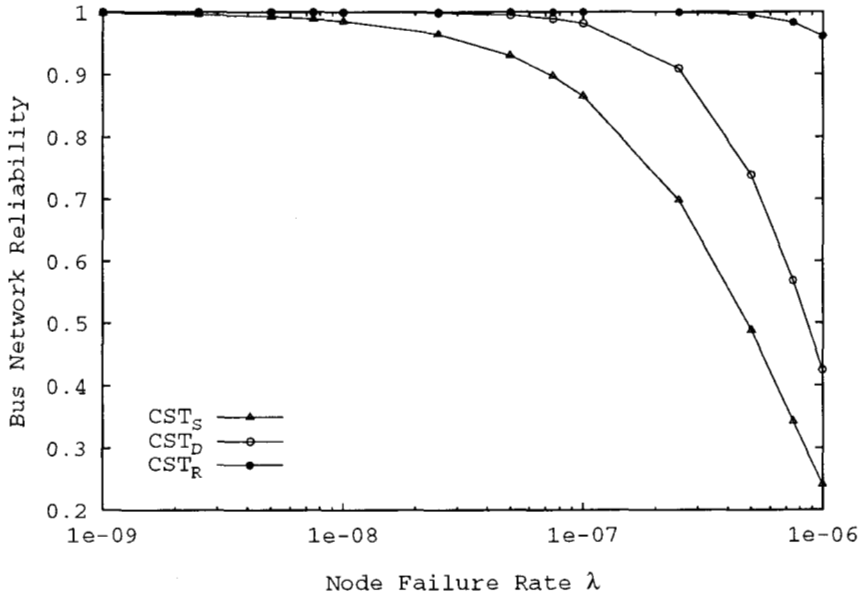

Figure 15: Bus Network Reliability as a Function of Node Failure Rate ($n = 16$)

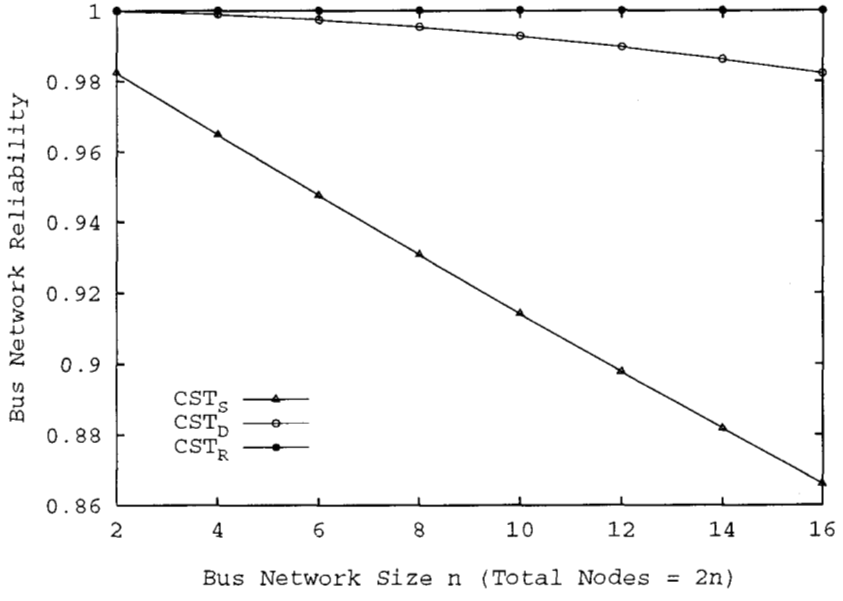

Figure 16: Bus Network Reliability as a Function of Network Size ($\lambda = 10^{-7}$)

Figure 17 illustrates the evaluation results of a study in which λ and n are set to 10^{-7} and 16, respectively, while mission duration t is kept as a variable parameter. It is clear that both $R_{\mathrm{S}}^{\mathrm{CST}}$ and $R_{\mathrm{D}}^{\mathrm{CST}}$ become unacceptable for long-life missions. On the other hand, $R_{\mathrm{R}}^{\mathrm{CST}}$ remains very reasonable even when $t = 100,000$ (about 11.5 years).

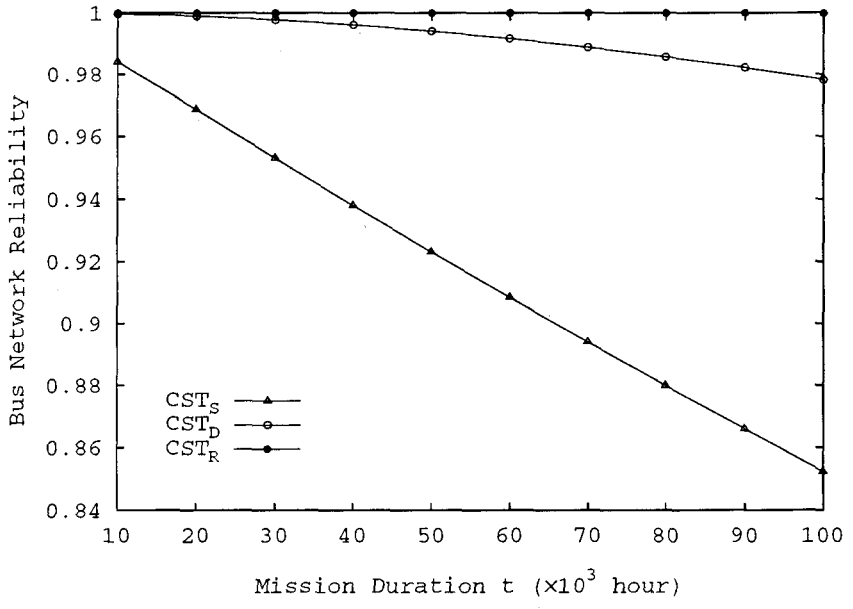

Figure 17: Bus Network Reliability as a Function of Mission Duration ($\lambda = 10^{-7}$)

6 Conclusion

To implement fault tolerance in a COTS-based system is becoming a major challenge today when cost concern has led to increased use of COTS products for critical applications. On the other hand, vendors remain reluctant to incorporate fault tolerance features into COTS products because doing so is likely to increase development and production costs and thus weaken the market competitiveness of their products. Therefore, to cope with the current state of COTS is crucial for us. Accordingly, the significance of our work reported in this paper is two folds:

1) Our experience demonstrates that thorough evaluation and innovative utilization of pertinent standard features of a state-of-the-practice COTS product could enable us to circumvent their shortcomings and facilitate effective implementation of a COTS-based fault-tolerant system for critical applications, and

2) Our design and the resulting system which is anticipated to to be delivered to deep-space missions in the near future may stimulate many other developments of COTS-based highly reliable systems, which in turn, could encourage the vendors to incorporate fault tolerance features as *implementation options* of COTS products. These features will permit a COTS product, in a cost-effective manner, to satisfy the customers in both critical and non-critical application areas.

How to provide fault tolerance for the I2C bus for protecting of the IEEE 1394 bus is beyond the scope of this paper. A number of such techniques have been developed at JPL CISM, which will be published in the near future. Currently, we are designing simulation methods to assess and validate the fault detection algorithms reported in this paper. Moreover, we are motivated to develop a paradigm that will provide guidelines for adopting COTS to space applications.

7 Acknowledgment

The authors wish to express their appreciation for Mr. William Charlan and Mr. Huy Luong at the Jet Propulsion Laboratory for their stimulating discussions and refreshing ideas. This work is performed by Jet Propulsion Laboratory, California Institute of Technology, and funded by NASAs X2000 Program.

References

[1] L. Alkalai, "NASA Center for Integrated Space Microsystems," in *Proceedings of Advanced Deep Space System Development Program Workshop on Advanced Spacecraft Technologies*, (Pasadena, CA), June 1997.

[2] L. Alkalai, "A roadmap for space microelectronics technology into the New Millennium," in *Proceedings of the 35th Space Congress*, (Cocoa Beach, FL), Apr. 1998.

[3] L. Alkalai and A. T. Tai, "Long-life deep-space applications," *IEEE Computer*, vol. 31, pp. 37–38, Apr. 1998.

[4] L. Alkalai and S. N. Chau, "Description of X2000 avionics program," in *Proceedings of the 3nd DARPA Fault-Tolerant Computing Workshop*, (Pasadena, CA), June 1998.

[5] T. Shanley and D. Anderson, *PCI System Architecture.* Addison Wesley, 1995.

[6] IEEE 1394, *Standard for a High Performance Serial Bus.* Institute of Electrical and Electronic Engineers, Jan. 1995.

[7] D. Anderson, *FireWire System Architecture, IEEE 1394.* PC System Architecture Series, MA: Addison Wesley, 1998.

[8] D. Paret and C. Fenger, *The I2C Bus: From Theory to Practice.* John Wiley, 1997.

[9] C. S. Raghavendra, A. Avižienis, and M. D. Ercegovac, "Fault tolerance in binary tree architecture," *IEEE Trans. Computers*, vol. C-33, pp. 568–572, June 1984.

[10] Y.-R. Leu and S.-Y. Kuo, "A fault-tolerant tree communication scheme for hypercube systems," *IEEE Trans. Computers*, vol. C-45, pp. 641–650, June 1996.

[11] IEEE P1394A, *Standard for a High Performance Serial Bus (Supplement), Draft 2.0.* Institute of Electrical and Electronic Engineers, Mar. 1998.

[12] K. Sasidhar, L. Alkalai, and A. Chatterjee, "Testing NASA's 3D-stack MCM space flight computer," *IEEE Design & Test of Computers*, vol. 15, pp. 44–55, July-September 1998.

[13] S. N. Chau *et al.*, "X2000 architecture tiger team meeting review," Technical Report, Jet Propulsion Laboratory, California Institute of Technoloty, Pasadena, CA, June 1998.